ENGR489 Final Report: Zero Harm Mobile App

Vincent Alvarez

I. INTRODUCTION

Upholding the Health and Safety of all workers on a work site is of the utmost importance to the management of the site. However, keeping track of personnel working on site, hazards, locations, and substances being used and keeping all site workers informed and up to date can be difficult. Having real time updates and communication between the workers on a site and the management is very difficult to achieve. For example, reporting an incident on a site can take quite some time as the location of the site must be mentioned, people involved, severity of incident and any hazards, substances or equipment involved in the incident. If updates about hazards, safety standards and emergency plans are not given to workers in real time, site workers' understanding of how to work safely on a site will become outdated. This can lead to an increased risk in safety on a site and can lead to site workers either hurting themselves or other workers.

A solution to this would be to create a mobile app which can be downloaded onto the workers' mobile phones. This app should allow the workers to:

- Induct workers into a site.
- Sign a worker in and out of a site.
- Read information about the site and its health and safety practices.
- Notify users of updates to safety and emergency plans.
- Report and Raise concerns on the site and report incidents that happen on the site.

This mobile app should be compatible with both an Android and iOS device and should be able to communicate with the already existing database used for the Zero Harm web app.

To ensure the mobile act work correctly the following features should be performed successfully:

- Workers can sign into a site using a QR code. This information is then persisted to the site management through the mobile app via the internet. This should be done in real time if a stable internet connection is available.
- A worker can induct themselves into a site also using a QR code. The QR code takes the worker to an induction page where they go through the process of inducting themselves into the site.
- A worker can access the site profiles of a site they have been inducted onto. This includes site information like address and management.
- A worker should be able to access a site's emergency contacts and plan and should be able to see a list of all hazards present on the site.
- Workers should be able to inform the management of an incident or hazard that happened on the site in real time. This incident report should be sent immediately to manage assuming internet connection is available.

Final Products Features

Induct and Sign In

The Final Product can induct a user into a site and sign them into a site. However, this is not done buy scanning a QR code but instead, instead the user must go to a site profile and induct. If the user has already been inducted, then the sign in option will be made available instead. Induction and sign in are done through a form and does not use QR code scanning.

Site Information

Information about a site can be viewed on its site profile. Site profiles of the sites a user is connected to is viewable on the dashboard of the mobile application. By clicking on the site profile information about that site will be displayed on the site profile page. This consists of general information like site address and health and safety plans, hazard information like hazard names, description and prevention methods and the emergency information about each site like emergency plans and contacts. All this information is readily available on each site profile displayed on the dashboard.

Notify Management

User can now send a message to management about concerns they see on the site. This is done through the notify button on each site profile page that, when clicked, brings them to a notify form. In this form the user can write a message they would like to send to the management and select a time in which the incident happened. This is not to be confused with an actual site report it is simply a message a worker can send to that site's management.

II. RELATED WORK

As part of the research for this app, research was done on the existing solutions for a similar issue. These apps mainly stem from work management software are where mobile apps developed for the worker to better communicate to their management through there software. The five main apps that research was done into was:

- Safe365
- HazardCo

- SignOnSite
- Site Docs
- Site App Pro

It is to be noted that when researching these apps, the focus was not on user experience but rather what features did these apps have what features they not have.

This table shows the different features that each app has.

Company	QR sign in	Induction	Access site Documents	Hazards lists	Reports	Offline use
Safe365	✓			✓	✓	
HazardCo	✓		✓		✓	
SignOnSite	✓	✓	✓			
Site Docs	✓	✓	✓			✓
Site App Pro	✓	✓		✓	✓	✓

From this table we see that many of the existing apps all support signing into a work site by scanning a QR code. Three out of the five apps also support inducting users into a site. From the table above we can see a clear lack of hazards listing on mobiles apps. Having the hazards of a site available for the user to see and having the site's emergency contacts and plans available to the worker is a focus of the Zero Harm Mobile App. This is so that workers have all information they need to work safely on a site within their phone, reducing the risk of accidents or incidents happening on the site.

All apps meet the basic requirement of letting the management of a site know who is and who is not currently working on their sites. These apps simply help management manage their personnel on site, but this design does not give the workers to communicate back to management. The Zero Harm Mobile App aims to put the power into the workers hands, allowing for open communication between workers and sites management. This will be done through allowing the user to report incidents directly to the management through the app. Only 3 of the 5 apps researched allowed for reports to be made from workers to management so this will be a focus for the app to help differentiate itself from other apps.

Tools and Methodology

The Zero Harm App will run both on Android and iOS devices. This means the app should be written to be compatible with both environments. To avoid having to write the same app in 2 different languages the app will be written in React Native. React Native is an open-source framework which allows for the creation of a mobile app which is compatible with both iOS and Android. This does this by interpreting either TypeScript or JavaScript language which defines the functionality of the app and the UI components and converts them into each environment's native language [1].

This streamlines the development as this framework allows for develop of both Android and iOS versions of the mobile application at the same time. React Native is open source meaning many different components already exist online which can be used in the Mobile App, keeping the development simpler. React Native allows for changes of code in the project to be updated to the app in real time [2]. This means that any changes that are made to the code base is automatically reflected in a running emulation of the mobile application in real time. This allows for faster feedback for changes made in the mobile application since changes are made in real time. This cuts back development time as the app does not have to be repackaged or recompiled every time a change is made to it.

There were several different libraries used in the development to this mobile application. These were the ones that contributed the most to the mobile application's functionality.

Apollo GraphOS

Apollo GraphOS is a library that allows for the use of GraphQL in both react and react native applications [1]. The API used to connect to the already existing Zero Harm database is a GraphQL API. This library has the app the ability to use GraphQL to connect to the database. This library allowed the mobile application to query [4] information about the user, the sites the user was a part of and allows the mobile application to make mutations [5] like Sign In and Out and notify management. This library benefitted the design of the mobile application because it allowed for easier execution of mutations and queries made through GraphQL API.

It also makes handling responses from the GraphQL queries and mutation easier as it makes use of hooks for each type of response the GraphQL query or mutation returns. Examples of these hooks include loading, a Boolean which indicates if the query or mutation is still loading and error, a Boolean that indicates if the query or mutation failed. All data return in the response is encapsulated as a hook called data. This benefitted the design of the app as I did not need to parse the body response of the query or mutation response, this was done by the Apollo client.

React Native Async Storage

Async storage is an asynchronous, unencrypted, persistent, key-value storage system for React Native [6]. This allows a react native mobile application to store string values asynchronously and data from this storage can be accessed at point in the mobile application or in any scope of the code [7]. This benefits the design of the app as variables like the session ID of the login user or the visitId used in the Sign In and Out mutations can be store using this feature. This means that these 2 values can be set and extracted in any scope of the mobile application.

React Native Cookies

React Native Cookies allows a react native app to handle cookies being used during its execution [8]. This library gives the mobile application the ability to set cookies, get cookies and clear cookies being used. This benefits the design of the mobile application because the cookies automatically handled by the react native environment needs to be cleared. The mobile application requires users to log into their Zero Harm account which returns a cookie for authentication. The React Native environment handles this cookie but based on the documentation React Native has some known issues regarding cookie-based authentication and its own environment [9].

Because of this the mobile application handles cookie-based authentication itself by manually extract the cookie from the header from the login response and clears all other cookies possibly stored anywhere else in the system by using React Native Cookies. Specifically, it uses clearAll() to remove all cookies currently being store. This is so that the cookie manually stored by the app and any cookies stored by the environment do not cause issue to one another.

React Native Navigation

React Native Navigation allows applications to smoothly transition from different screens established in the application [10]. The main feature used from stack navigator and bottom tab navigator [11, 12]. Stack navigator allows for transition between screens of the application and keeps the history of the route the user takes as a stack. Bottom Tab navigation can create a bottom tab bar at the bottom of a screen. These tabs can then be clicked to take you to different screens. This benefits the design of the mobile application because the path in which the user goes down can be determined by the navigation tool. For example, if a user's login is successful the app can transition the user to the home page screen by using the navigate function.

Another example is when user is in the dashboard, goes into a site profile and now wants to go back to the dashboard. The stack navigator knows which screen the user was on previously so by simply using the native back functionality (gesture swipe back (iOS or android) or back button (Android)) the user can be brought back to the page they were on previously.

III. DESIGN

Conceptual Design

The mobile application uses existing resources that are used in the existing Zero Harm web app. This web app has an API written in GraphQL communicates changes made by the user on the web app to the Zero Harm database. The API also helps the web app retrieve the relevant information from the database to display for the user e.g., site information. This same structure is used in the Zero Harm mobile app, using the existing GraphQL API to communicate changes from the mobile app to the database and retrieve the relevant data to display to the user. The structure of the app can be broken down into 3 different sections: application section, GraphQL section, database section.

Application Section

This section is essentially the mobile app UI itself. Here the user interacts with the application meaning 100% of all user interaction will happen in this section. User inputs are collected in this section i.e., incident reporting and signing in and out of a site and then use the GraphQL section to send this information to the database.

GraphQL Section

This section sits in between the application section and the database section. This section communicates between the database and application section pulling information from one end to another. Any information that needs to be pushed from the mobile app to the management of the site will be handled by this section. Any information the mobile app needs from the database will be queried by the GraphQL section and given to the application section. The application section and database section never directly communicate with each other, all communication is handled by this section.

Database Section

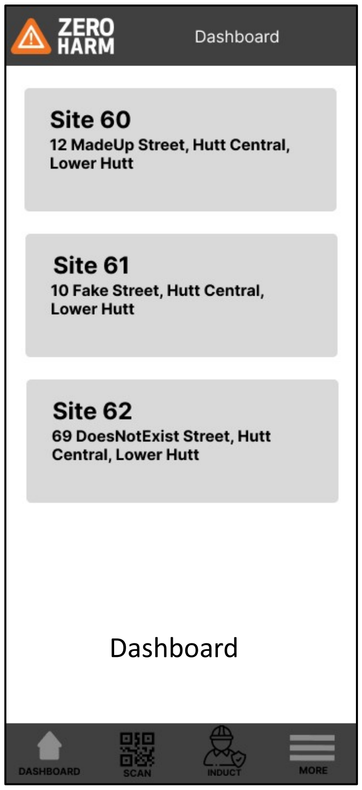
This section contains the database used in the existing Zero Harm web app. This contains all information about the sites registered in Zero Harm, the management and admins registered in Zero Harm and all workers registered in Zero Harm. All data being represented in the application section will be from this database and all information pushed from the application section will be stored and sorted in this database. Any information concerning reports or signing in and out of a site that is pushed to the database will then be relayed to management.

The reason this structure was considered was so that the mobile application's functionalities and the GraphQL functionalities are not coupled together i.e., the mobile functionalities are independent of the GraphQL functionalities. This is so that changes made to the mobile application GraphQL API did not have a significant impact on the implementation of the GraphQL API. This was also done because changes and upgrades are constantly being made to the GraphQL API by Site Safe. By keeping the Application section separate from the GraphQL section the Application section would not be heavily effect by changes made in the GraphQL section. It also meant that there was more freedom when designing how information was parsed and displayed into the application.

Wireframes

Wireframes of the basic layout of the map including a dashboard screen and site profile screen were made to better understand what exactly the layout of the map would look like and what information will be needed from the API.

Examples of the wireframes created are on the next page:



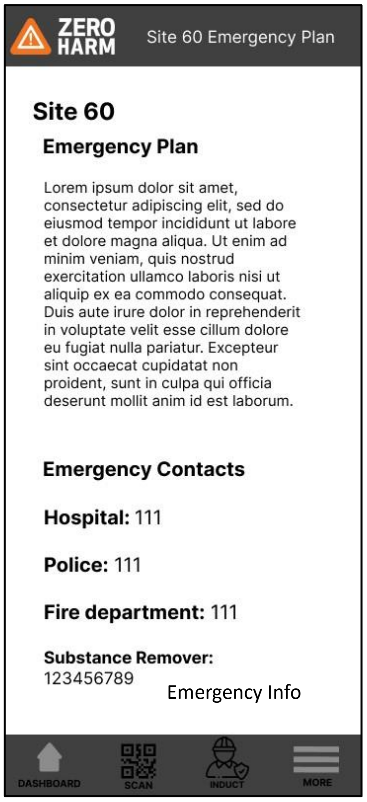
Site 60
12 MadeUp Street, Hutt Central, Lower Hutt

Site 61
10 Fake Street, Hutt Central, Lower Hutt

Site 62
69 DoesNotExist Street, Hutt Central, Lower Hutt

Dashboard

Figure 1



Site 60

Emergency Plan

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Emergency Contacts

Hospital: 111

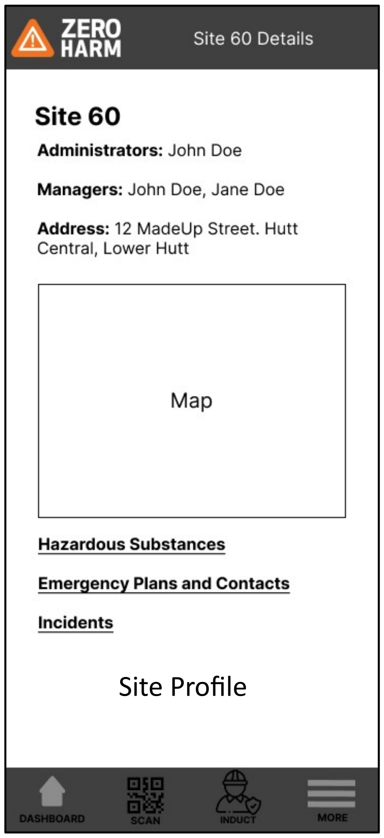
Police: 111

Fire department: 111

Substance Remover: 123456789

Emergency Info

Figure 3



Site 60

Administrators: John Doe

Managers: John Doe, Jane Doe

Address: 12 MadeUp Street. Hutt Central, Lower Hutt

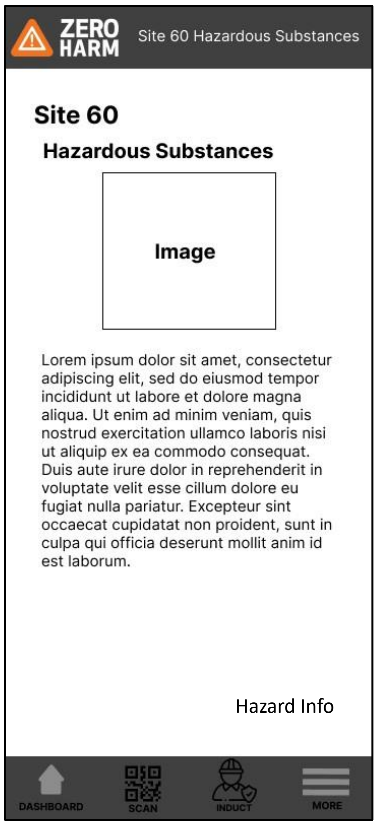
Hazardous Substances

Emergency Plans and Contacts

Incidents

Site Profile

Figure 2



Site 60

Hazardous Substances

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Hazard Info

Figure 4

Sustainability

The design of this mobile application addresses the 3rd goal of sustainability of good health and wellbeing [16]. When using this app on site workers will have the resources needed to conduct themselves safer on site within the app. This includes having access to site hazards and being able to report hazards seen on site. With this, overall risk can be decreased on the site ensuring a healthier and safer work environment. This creates help maintain a worker's healthier life by reducing chances of injuries to themselves and others.

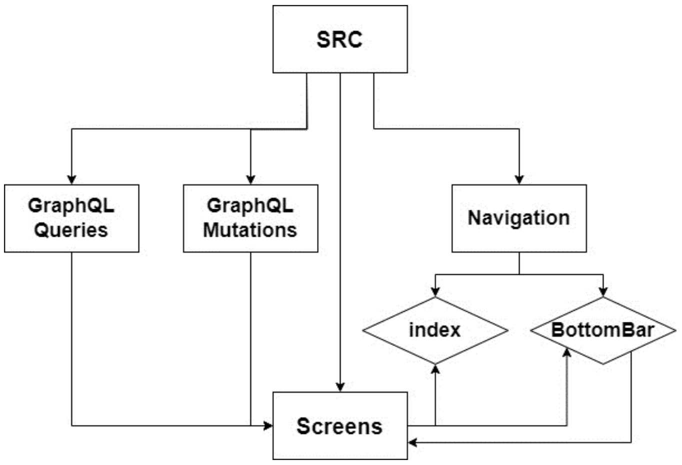
IV. IMPLEMENTATION

When designing the structure of the mobile application the directories were broken down into four main parts:

- Screens Directory
- Navigation Directory
- GraphQL Mutations Directory
- GraphQL Queries Directory

This is done so that finding and identifying different components can be easier for development and to make each functional part of the mobile application modular from one another. This is so that specific queries and mutations weren't so dependent on specific screens e.g., the site query is not dependent or closely related to the site profile screen. From here components in GraphQL Mutations, GraphQL Queries are used in the Screen directory so that the screen components can call queries and use the mutations needed to sign in and out of a site and get the user's site information.

Navigation Directory



A visualisation of the directory structure of the Zero Harm Mobile App. The rectangles are directories, and the diamonds are components in the directories.

All components used in the Navigation directory make use of the React Navigation library [10].

The navigation component, stored in the file called index.tsx, isn't explicitly used any other directory. It is there to set up the stack navigation path of the application by creating a stack navigator. When a stack navigator is created you must input stack screens into it which consists of screens components. The screen components used in the stack navigator of the mobile come from the screen component meaning the screens created for the mobile application are used in the navigation stack. Transition between each screen in the stack navigator is determined the navigate function made possible with the use of the useNavigation hook. Transition between screens in the stack navigator done by calling the navigate function [15].

The bottom bar component is also defined within the navigation directory. This creates a bottom bar navigation tab which is displayed on the home screen component in the Screens directory. This bottom bar component makes use of the screens already established in the Screens directory, specifically the dashboard and more screen.

ApolloProvider

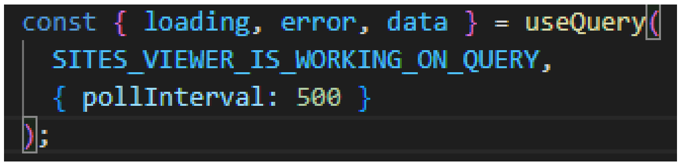
The entire mobile application is wrapped in ApolloProvider component [13]. The way this is done is by encapsulating every component being returned by the App function with ApolloProvider tags. The App function is stored in a typescript file called App stored outside of the src directory. By wrapping the entire mobile application with these tags, you can use Apollo's useQuery [4] and useMutation [5] hooks at any point or scope in the application.

GraphQL Queries Directory

The GraphQL Queries Directory contains all the files which create and export the different queries used throughout the app. All queries created in this directory were done using GraphQL [3]. The 2 queries that are stored in this directory, the SiteProfileQuery and SitesViewerIsWorkingOn query.

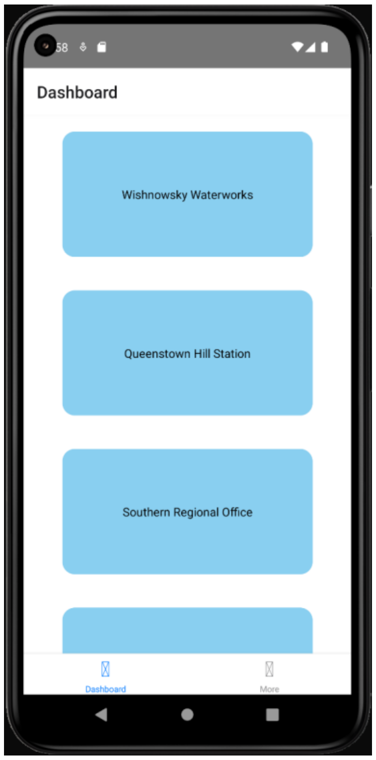
SitesViewerIsWorkingOn is stored in a TypeScript of the same name in the GraphQL Queries directory. The query used is called SitesViewerIsWorkingOn, a query defined in the schema of Site Safe's GraphQL API. This query pulls information about any of the sites that the logged in user is currently working on and returns the information in its response. If the query is successful, parses the data received into an array called siteData. siteData is a variable used in the dashboard page to display the sites on the page. The information specifically pulled in this query is the Site ID, name, and description. The reason as to why it pulls this specific information is because its response data is used in the dashboard. The dashboard does not display site profile information, it only displays the number of sites the user is working on as rectangles on the screen:

The method used to pull the relevant information is the useQuery method, a React hook which can be used via the Apollo Client library and the Apollo Provider. The useQuery method can take in 2 parameters a query parameter and an options parameter. The query parameter is the query itself which is written in string then parsed by a function called gql which converts the string into a query document [4]. The options parameter allows for variables to be inserted into the query and other operations that can be done on the query [4]. For this specific instance the options parameter was used to insert a polling time of 500 milliseconds. By inserting a polling time option into useQuery, the query passed into the function will refetch its information every 500 milliseconds. This means that the information being displayed is constantly being refreshed. The time interval of polling time can be any value in milliseconds, I chose 500.



```
const { loading, error, data } = useQuery(
  SITES_VIEWER_IS_WORKING_ON_QUERY,
  { pollInterval: 500 }
);
```

Example of useQuery in SitesViewerIsWorkingOn.tsx



An example of what the dashboard displays.

SiteProfileQuery is stored in a TypeScript file of the same name in the GraphQL Queries directory. This file not only queries a site's information but also returns the site profile of the site. This file is used by the Site Profile screen in the Screens directory to display the profile of the site the user clicked on in the dashboard screen. SiteProfileQuery makes use of the useQuery hook provided by Apollo client and Apollo Provider. It uses the query called Sites; a query defined in Site Safe's schema for their GraphQL API. This query takes in a Site ID as a parameter and returns information of the site corresponding with that Site ID. In SiteProfileQuery, useQuery runs this query with a written version of the query parsed by a gql function, Site ID value which will be inserted into the query, and a polling time of 500 milliseconds as parameters. This means all information on a site profile is refreshed constantly, meaning it is up to date. The information returned by this query is all the displayable information about the site.

Once a response has been received the data is then parsed into an object called siteDetails. SiteQueryProfile, while using siteDetails, constructs the entire site profile page of the site it just query. This includes creating tabs which display the sites hazard and emergency information and display the buttons used to induct, sign in, sign out and notify.

To be clear, the SiteProfileScreen file attach the navigation stack does not do any query calling or construction of the site profile components, it simply imports the file SiteProfileQuery and runs it within itself. By running SiteQueryProfile, it gets the actual site profile components from SiteQueryProfile and displays it on SiteProfileScreen.

Whether the sign in, sign out or induct button is displayed on the site profile is determined by viewerCanSignIn, viewerCanSignOut, and viewerCanInduct Booleans which are queried from the useQuery function call. For example, if viewerCanSignIn is true then the sign in button is displayed, if viewerCanSignOut is true then the sign out button is displayed, and of viewerCanInduct is true then the induct button is displayed.

GraphQL Mutations Directory

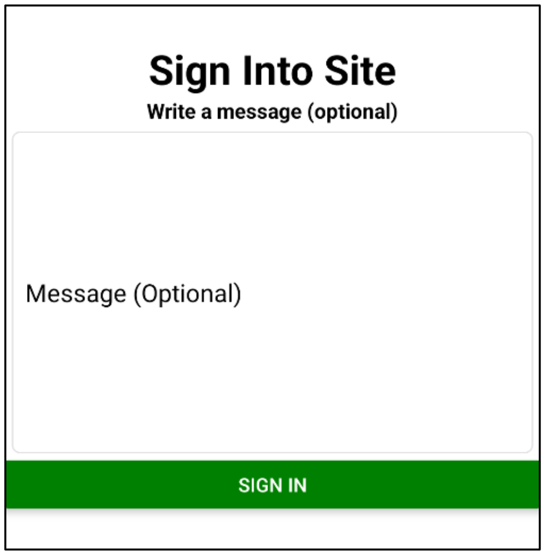
GraphQL Mutations contains all files which have to do with mutations. Mutations in GraphQL allows for the modification of existing data or the creation of new data in a database. All mutations used in this directory are strictly used in the SiteProfileQuery components that are turned by that that file. The mutations stored in this directory are:

- SignIntoSite Mutation
- SignOutSite Mutation
- Notify Mutation
- InductIntoSite

SignIntoSite is stored in a TypeScript file of the same name in the GraphQL Mutations directory. This file returns the form the user uses to sign into a site and, once the form is completed, runs the mutation to sign in a user into a site. The form is made visible when the Sign In Button is pressed in the site profile page. The form consists of a dialog box where a user can write a message and a sign in button which runs the mutation. The mutation used in this file was the signIntoSite mutation defined in Site Safe's GraphQL API schema. The mutation reflects this change in the database and creates a visit based off the sign in and returns the visit ID of the newly made visit. This is then stored into async storage for the sign out mutation to use.

The function used to execute the mutation is useMutation, a React hook from the Apollo client library and made functional with the use of Apollo Provider. useMutation, like useQuery, takes in two parameters: the mutation which is written in string then transformed into a query document use the gql query, and the options parameter which can be variables to be inserted into the mutation or any other mutation option available. In SignIntoSite, the parameters passed in were the mutation to sign into site and the variables that needed to be inserted into the mutation. These variables included the ID of the site the user was signing in to, an optional message the user could attach to the sign in, and a list of hazard IDs that the user had acknowledged when signing in. This information is passed into SignIntoSite by SiteProfileQuery when a user clicks submit when filling out a sign in form.

SignIntoSite creates the sign in form components and returns them to SiteProfileQuery for it to render when the sign in button is clicked.



Sign In form generated by SignIntoSite

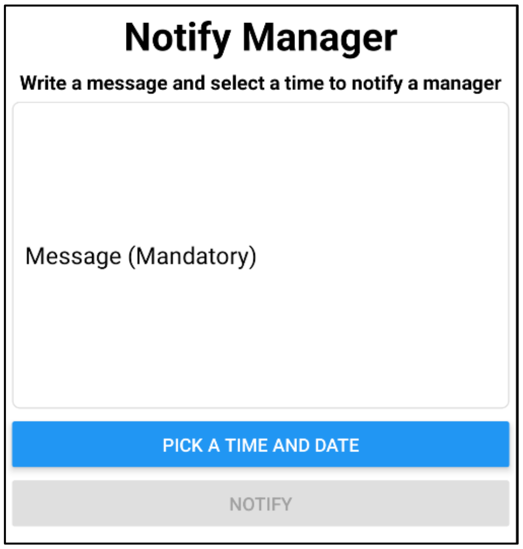
SignOutSite is stored in a TypeScript file of the same name in the GraphQL Mutations directory. This file returns the form containing a button which, when pressed, signs a user out of the site. The form is made visible when the Sign Out Button is pressed in the site profile page. The form only contains a sign out button. The mutation used in this file was the signOutSite mutation defined in Site Safe's GraphQL API schema. The mutation reflects this change in the database by cancelling the current valid visit the user has with the site. It does this by passing the visit ID the application has in async storage into the mutation.

SignOutSite uses the useMutation React hook to run execute the mutation once the user has clicked the sign out button. This useMutation has the mutation itself and the visit ID as arguments passed into it to when executed.

Notify is stored in a TypeScript file of the same name in the GraphQL Mutations directory. This file generates the form component used to create a notify message and runs the mutation to create a notify message in the database. This file uses the sendConcern mutation defined in Site Safe's GraphQL API schema. The form generated by Notify consists of a dialog box where a user can type out their message they would like to their management and a button which, when clicked, allows users to select a time as to when their concern occurred.

The mutation can only be executed if both a time and message value have been give. Once given, both the time and message values are extracted from the form and passed into the useMutation function as variables to be inserted into the already established sendConcern Mutation. The useMutation function is then executed, creating and sending a concern to the management of that site.

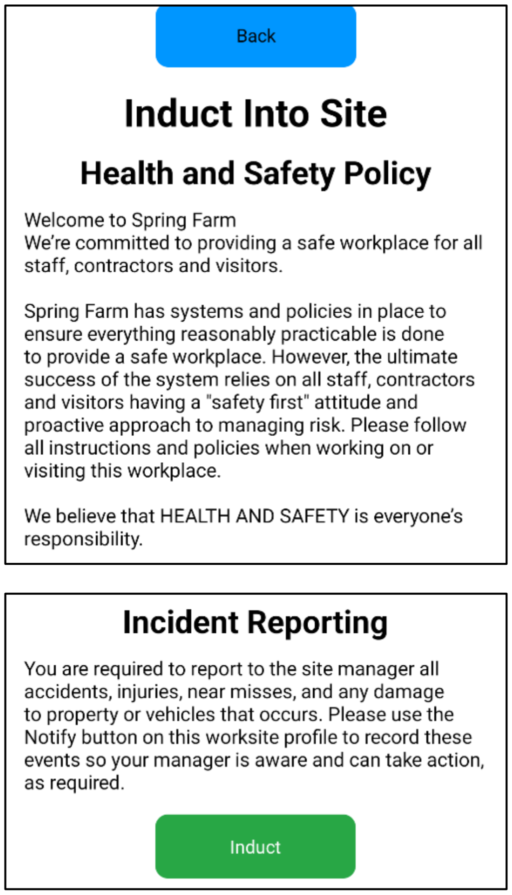
Notify creates the notify form components and returns them to SiteProfileQuery for it to render when the notify button is clicked.



Notify form generated by Notify.

InductIntoSite is stored in a TypeScript file of the same name in the GraphQL Mutations directory. This file generates the induction page and runs the mutation that inducts a user into a site in the database. The mutation used is then induct mutation which is defined in Site Safe's GraphQL API schema. The induction page generated by InductIntoSite consists of the health and safety policy, hazards, emergency plana and incident reporting information of a site. At the bottom of all this is the button which allows a user to induct into a site. This information is passed to InductIntoSite by SiteProfileQuery when it renders this page in the site profile components it generates.

InductIntoSite creates the induction page components and returns them to SiteProfileQuery for it to render when the induction button is clicked.



Induct Into Site

Health and Safety Policy

Welcome to Spring Farm
We're committed to providing a safe workplace for all staff, contractors and visitors.

Spring Farm has systems and policies in place to ensure everything reasonably practicable is done to provide a safe workplace. However, the ultimate success of the system relies on all staff, contractors and visitors having a "safety first" attitude and proactive approach to managing risk. Please follow all instructions and policies when working on or visiting this workplace.

We believe that HEALTH AND SAFETY is everyone's responsibility.

Incident Reporting

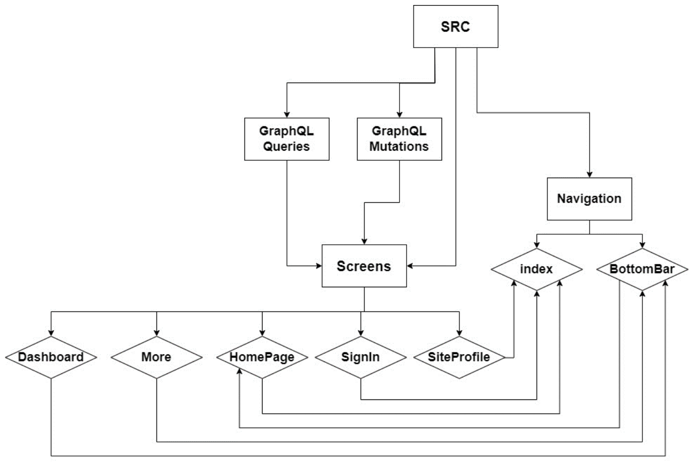
You are required to report to the site manager all accidents, injuries, near misses, and any damage to property or vehicles that occurs. Please use the Notify button on this worksite profile to record these events so your manager is aware and can take action, as required.

Portions of the Induction form generated by InductIntoSite.

This follows the design set out before as it separates the GraphQL components separate from the screen and navigation components. This means that the navigation and screen displays operate completely independently from the GraphQL components.

It also satisfies the requirements as these mutations allow the user to sign in and out of a site, induct into a site, and notify management about events on the site. Through the queries a user can see all sites they are working on and can also find all site information about each site. This information

Screen Directory



Structure of the screens in relation to navigation. Rectangles are directories and diamonds are components.

The screen directory consists of all the screens used in the mobile application. These screens consist of:

- Sign In Screen
- Home Page
- Dashboard
- More Page
- Site Profile

This follows the design as these screens are completely independent of the GraphQL components of the mobile application. The functionality of the screen components can function independently of the GraphQL components.

The screens can be split into two groups: stack navigator group, the screens being used in the stack navigator, and bottom bar navigator group, the screens being used in the tab navigator. The screens in the stack navigator are screens that can only be transitioned using when the navigate function is called. For each screen the navigate function is called in different places e.g., navigate is called in Sign In Screen of the response to the login attempt is 200. Screens used in the bottom bar screens can be accessed by clicking the different icons in the bottom bar i.e., when dashboard is clicked then the user the user is taken to the dashboard page.

Sign In Screen

The sign in screen is the first screen the user sees when the mobile application is booted. This is where the user can enter their Zero Harm account email and password to log into their account. This page uses no GraphQL mutations or Apollo libraries. All login in procedures is handled using the fetch function which sends a HTTP request to the Zero Harm servers. Requests are handled by the file itself by reading the response code return in the header. If the response code is 200, the file uses the navigate function to transition the user to the home screen page.

Home Page Screen

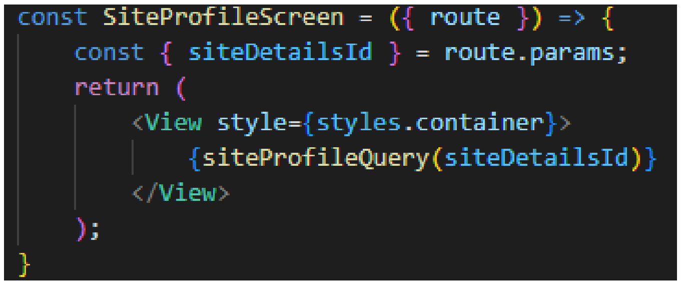
The home screen page is unique from the other pages because its purpose is to display the bottom tab component. Effectively, the home page screen allows for the bottom tab navigator to be nested inside of the stack navigator, allowing there to be the use of 2 navigators in one application. When the bottom bar navigator is returned by home screen the default page of the bottom bar navigator is displayed. In this case, the default screen is the dashboard screen.

Dashboard Screen

The dashboard screen is the first screen displayed by the Home Page Screen and shows all sites the user is currently working on. This is done by importing the SitesViewerIsWorkingOn component from the queries folder and returning the components return by the query component. Any time the user clicks on a site displayed on the dashboard the dashboard screen finds and extracts the ID of that site from the siteData array and passes it to the Site Profile Page. It then navigates to the site profile page using the navigate function.

Site Profile Screen

Site Profile Screen imports the SiteProfileQuery and uses the component to generate the site profile. It does this by the taking the site ID that the dashboard page passed to it and giving it to the SiteProfileQuery component as an argument in its function call.



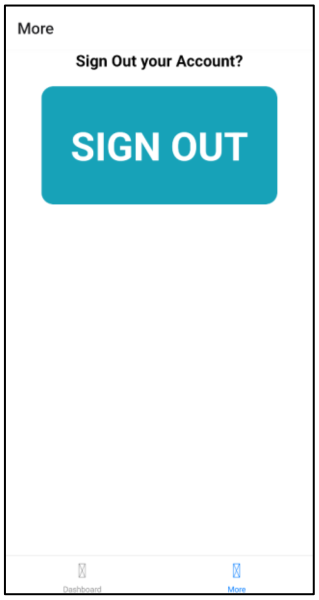
```
const SiteProfileScreen = ({ route }) => {
    const { siteDetailsId } = route.params;
    return (
        <View style={styles.container}>
            {siteProfileQuery(siteDetailsId)}
        </View>
    );
}
```

Example of site profile page giving the site ID to SiteProfileQuery.

The Site Profile Screen then renders the site profile screen returned by SiteProfileQuery.

More Page Screen

The more page is rendered by the bottom bar component and is only ever made visible when its tab is pressed by the user. This page is rendered on the Home Page Screen and displays the log out button. This page was implemented to simply allow the user to log out of their Zero Harm account. This button does not use any GraphQL or Apollo client but rather uses HTTP request to log out of the application.



Log Out Screen displayed on the More Page Screen.

V. EVALUATION

To evaluate the mobile application the application was testing on real workers on a working site. Because the participants for these user tests were not students at the university an ethics application had to written and approved by the university. This resulted to recording the user's consent instead of collecting the user consent via forms.

To evaluate the usability of the mobile application user tests needed to be conducted on real site workers to see if the mobile application was usable from the perspective of its target audience, the site workers themselves. Gathering feedback from site workers can give a better understanding of whether the mobile app in its current state is understandable and easy to use or if the application is not at all usable.

Evaluation Method

To properly test the application a user test script was created. This script would be used to test each participant of the user test so that each user test was consistent. The user tests went as followed:

1. Participants would be asked if they consent to being a part of the user test and they consent to their test being recorded. Their response was video recorded for evidence of consent.
2. Participants would be given a mobile phone with the app installed, already on the dashboard of a dummy user.
3. Participants are asked to complete specific tasks by the interviewer with limited instruction from the interviewer.
4. After each task the user is asked to rate the difficult of each task. If the task was very difficult the interviewer would ask why.

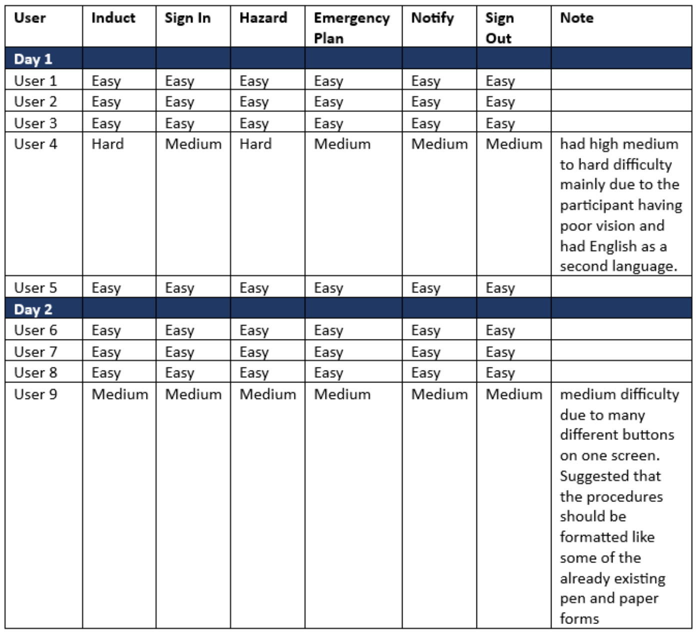
This user test was conducted on 15 different participants and for every task they found difficult they were asking as to why it was difficult. The focus of the user testing was to get the overall opinion about the mobile application from site workers and to see if site workers found it easy or difficult to use the application.

Each task in the user testing script covered the use of an essential feature of the application. These features included:

- Induction.
- Sign into site.
- Viewing hazards.
- Viewing emergency information.
- Notifying Management.
- Sign out of site.

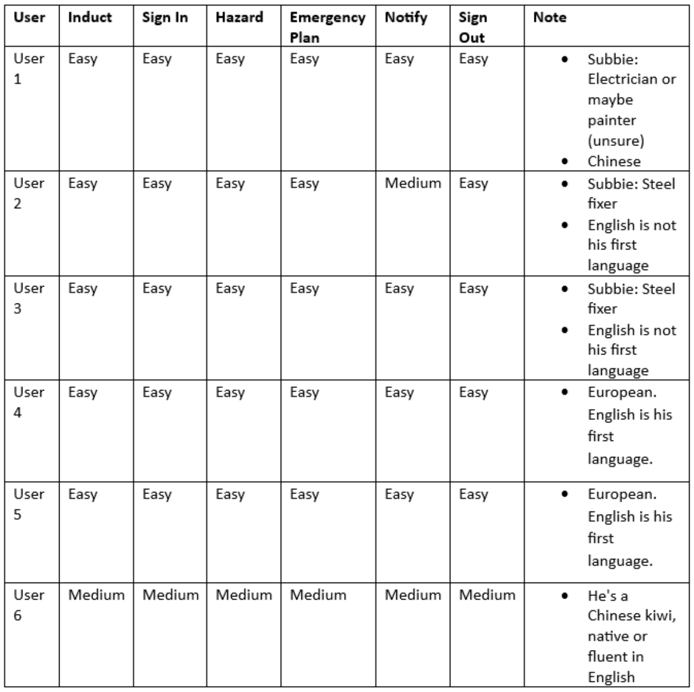
This was to ensure that each participant had an opportunity to use these features and comment on whether it was easy or hard to use the feature. After every task the user was asked to rate the difficulty of each task between easy, medium, and hard. Keeping the ratings of each task simple and within 3 different options was intentional. This was done so that the rating system would not overwhelm any of the participants especially since some of the participants first language is not English.

Results



User	Induct	Sign In	Hazard	Emergency Plan	Notify	Sign Out	Note
Day 1							
User 1	Easy	Easy	Easy	Easy	Easy	Easy	
User 2	Easy	Easy	Easy	Easy	Easy	Easy	
User 3	Easy	Easy	Easy	Easy	Easy	Easy	
User 4	Hard	Medium	Hard	Medium	Medium	Medium	had high medium to hard difficulty mainly due to the participant having poor vision and had English as a second language.
User 5	Easy	Easy	Easy	Easy	Easy	Easy	
Day 2							
User 6	Easy	Easy	Easy	Easy	Easy	Easy	
User 7	Easy	Easy	Easy	Easy	Easy	Easy	
User 8	Easy	Easy	Easy	Easy	Easy	Easy	
User 9	Medium	Medium	Medium	Medium	Medium	Medium	medium difficulty due to many different buttons on one screen. Suggested that the procedures should be formatted like some of the already existing pen and paper forms

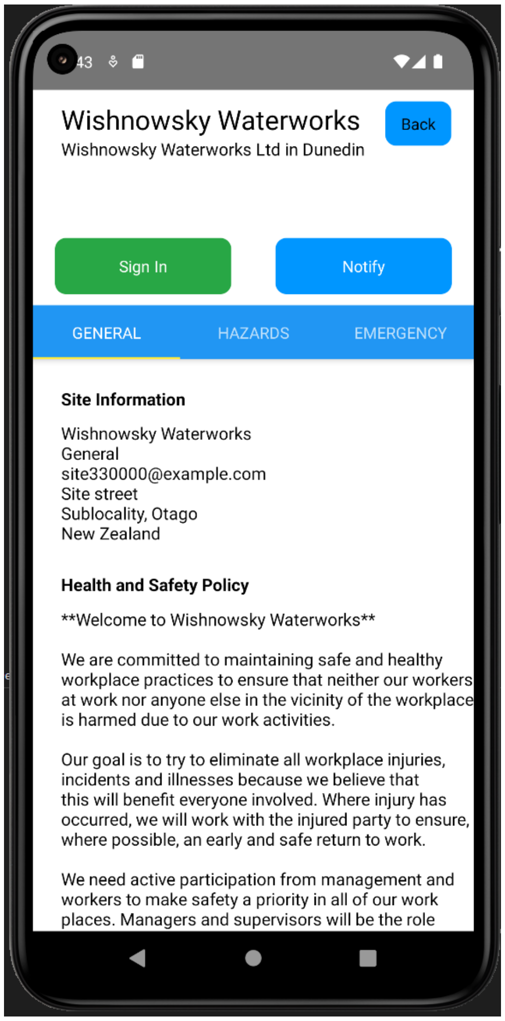
Figure 1



User	Induct	Sign In	Hazard	Emergency Plan	Notify	Sign Out	Note
User 1	Easy	Easy	Easy	Easy	Easy	Easy	• Subbie: Electrician or maybe painter (unsure) • Chinese
User 2	Easy	Easy	Easy	Easy	Medium	Easy	• Subbie: Steel fixer • English is not his first language
User 3	Easy	Easy	Easy	Easy	Easy	Easy	• Subbie: Steel fixer • English is not his first language
User 4	Easy	Easy	Easy	Easy	Easy	Easy	• European. English is his first language.
User 5	Easy	Easy	Easy	Easy	Easy	Easy	• European. English is his first language.
User 6	Medium	Medium	Medium	Medium	Medium	Medium	• He's a Chinese kiwi, native or fluent in English

Figure 2

As showed by the result many participants in the user tests found the mobile application easy to use. This could be mainly attributed to the fact that many of the participants were already proficient at English and that the UI layout of the site profile page made it obvious as to where to find information about the site and what button to press to perform a specific task.



Site Information

Wishnowsky Waterworks
General
site330000@example.com
Site street
Sublocality, Otago
New Zealand

Health and Safety Policy

Welcome to Wishnowsky Waterworks

We are committed to maintaining safe and healthy workplace practices to ensure that neither our workers at work nor anyone else in the vicinity of the workplace is harmed due to our work activities.

Our goal is to try to eliminate all workplace injuries, incidents and illnesses because we believe that this will benefit everyone involved. Where injury has occurred, we will work with the injured party to ensure, where possible, an early and safe return to work.

We need active participation from management and workers to make safety a priority in all of our work places. Managers and supervisors will be the role

Example of the site profile page.

The main reason as to why some of the users found the application hard to use was because a language barrier. Some participants did not have English as their first language meaning understanding the text of the application was harder to understand.

VI. FUTURE WORK

The Zero Harm Mobile App is clearly not in its final iteration and the one presented here is just a simple prototype. QR code scanning for induction and signing into a work site should be considered. This was the original intention of this version of the application but due to time constraints, the feature was cut from development. The use of a map in the notify form to show where the incident or concern is that they are notifying about. This would better reflect the web application version of the Zero Harm app as that is a feature on that version. Being about to change the default language of the application to a desired language. This way the application would be easier to use and more accessible to users who do not have English as a first language.

Although the application was created in the React Native framework the application was never tested for an iOS device, strictly on an android device. There is some ambiguity regarding whether the mobile application does work on an iOS device although theoretically it should. Testing the application on iOS device would be the next step forward. One possible feature would also be to add geo sign in meaning that the application would track your location and as soon as it detects you are on site via GPS, the application signs you into the site.

REFRENCES

[1] “Core Components and Native Components · React Native,” *reactnative.dev*. https://reactnative.dev/docs/intro-react-native-components

[2] “Fast Refresh · React Native,” *reactnative.dev*, Jan. 12, 2023. https://reactnative.dev/docs/fast-refresh.

[3] “Apollo Docs Home,” *Apollo GraphQL Docs*. https://www.apollographql.com/docs/

[4] “Queries,” *Apollo Docs*. https://www.apollographql.com/docs/react/data/queries

[5] “Mutations in Apollo Client,” *Apollo Docs*. https://www.apollographql.com/docs/react/data/mutations/

[6] “react-native-async-storage/async-storage,” *GitHub*, Apr. 12, 2021. https://github.com/react-native-async-storage/async-storage

[7] “Usage | Async Storage,” *react-native-async-storage.github.io*. https://react-native-async-storage.github.io/async-storage/docs/usage

[8] “React Native Cookies - A Cookie Manager for React Native,” *GitHub*, Oct. 12, 2023. https://github.com/react-native-cookies/cookies/

[9] “Networking · React Native,” *reactnative.dev*, Jun. 21, 2023. https://reactnative.dev/docs/network#known-issues-with-fetch-and-cookie-based-authentication

[10] *Reactnavigation.org*, 2021. https://reactnavigation.org/docs/getting-started/

[11] “createStackNavigator | React Navigation,” *reactnavigation.org*. https://reactnavigation.org/docs/stack-navigator/

[12] “React Navigation,” *reactnavigation.org*. https://reactnavigation.org/docs/upgrading-from-5.x/#bottom-tab-navigator

[13] “Integrating with React Native,” *Apollo Docs*. https://www.apollographql.com/docs/react/integrations/react-native/

[14] “Queries and Mutations | GraphQL,” *graphql.org*. https://graphql.org/learn/queries/

[15] *Reactnavigation.org*, 2022. https://reactnavigation.org/docs/navigating/

[16] United Nations, “Goal 3: Ensure healthy lives and promote well-being for all at all ages,” *United Nations*, 2022. https://sdgs.un.org/goals/goal3